

Sending Broadcast Notifications with WebSocket, Spring Boot, React, STOMP, and SockJS

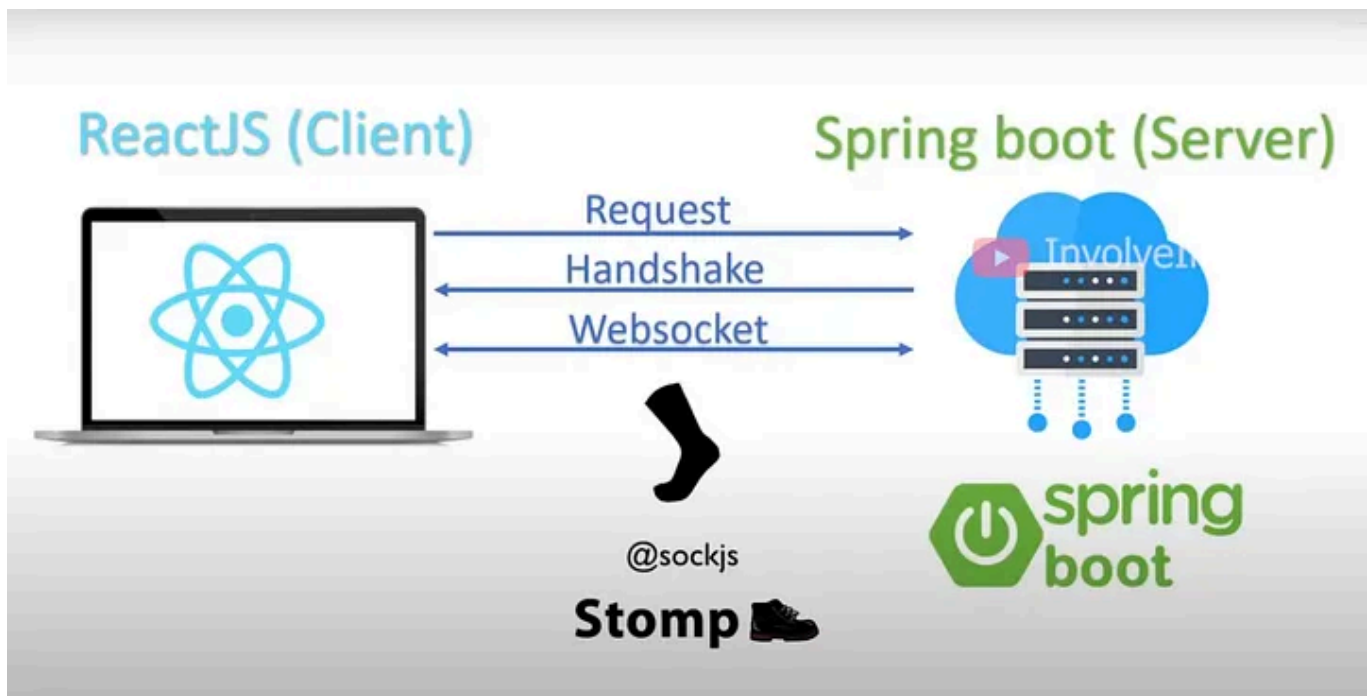


Deshani Palliyaguruge · [Follow](#)

4 min read · Jul 8, 2024



3



Hey there! Today, I'm going to share how you can implement real-time notifications in your web application using WebSocket, React, STOMP client,

SockJS, and Spring Boot. Real-time notifications are super important for any modern web app, and I'll show you how to handle broadcast notifications with examples. Let's dive in!

What Are Real-time Notifications?

Real-time notifications are alerts or messages that appear instantly on the user's screen without the need to refresh the page. They are essential for creating dynamic and responsive web applications. For example, when someone sends you a message on Facebook, you get a notification immediately without reloading the page. That's a real-time notification!

Setting Up the Project

To get started, we need to set up both the backend and frontend of our project. Here are the tools we will use:

- **Spring Boot** for the backend
- **WebSocket** for real-time communication
- **SockJS** for WebSocket fallback options
- **STOMP** (Simple Text Oriented Messaging Protocol) for messaging
- **React** for the frontend

Backend with Spring Boot

First, let's set up the Spring Boot application. We need to add dependencies for WebSocket and STOMP.

Step 1: Adding Dependencies

In your `pom.xml`, add the following dependencies:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-websocket</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter</artifactId>
</dependency>
```

Step 2: WebSocket Configuration

Next, create a configuration class to set up WebSocket and STOMP.

WebSocketConfig.java

```
package com.example.demo.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.messaging.simp.config.MessageBrokerRegistry;
import org.springframework.web.socket.config.annotation.EnableWebSocketMessageBr
import org.springframework.web.socket.config.annotation.StompEndpointRegistry;
import org.springframework.web.socket.config.annotation.WebSocketMessageBrokerCo

@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig implements WebSocketMessageBrokerConfigurer {

    @Override
    public void configureMessageBroker(MessageBrokerRegistry config) {
        config.setApplicationDestinationPrefixes("/app");
        config.enableSimpleBroker("/topic");
    }

    @Override
    public void registerStompEndpoints(StompEndpointRegistry registry) {
        registry.addEndpoint("/ws")
            .setAllowedOrigins("http://localhost:3000") // Allow requests fr
            .withSockJS();
    }
}
```

```
}  
}
```

In this configuration file:

- We enable WebSocket message handling by annotating the class with `@EnableWebSocketMessageBroker`.
- We configure a message broker with a prefix `/app` for messages sent from the client to the server and `/topic` for messages sent from the server to the clients.
- We register an endpoint `/ws` that will use SockJS for fallback options.

Step 3: WebSocket Controller

Next, we need a controller to handle WebSocket messages.

WebSocketController.java

```
package com.example.demo.controller;  
  
import org.springframework.messaging.handler.annotation.MessageMapping;  
import org.springframework.messaging.handler.annotation.SendTo;  
import org.springframework.stereotype.Controller;  
import com.example.demo.model.Greeting;  
  
@Controller  
public class WebSocketController {  
  
    @MessageMapping("/hello")  
    @SendTo("/topic/greetings")  
    public Greeting greeting(String name) {  
        return new Greeting("Hello, " + name + "!");  
    }  
}
```

```
}  
}
```

In this controller:

- The `@RequestMapping("/hello")` annotation maps messages sent to `/app/hello` to the `greeting` method.
- The `@SendTo("/topic/greetings")` annotation ensures the returned `Greeting` object is sent to all subscribers of the `/topic/greetings` topic.

Step 4: Greeting Model

We need a simple model to represent the greeting message.

Greeting.java

```
package com.example.demo.model;  
  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
public class Greeting {  
    private String content;  
}
```

Step 5: React Frontend

To set up the React frontend to interact with our WebSocket server, follow these steps:

1. Create a React App: If you don't have a React app set up already, you can create one using Create React App. Open your terminal and run:

```
npx create-react-app websocket-client  
cd websocket-client
```

2. Install STOMP and SockJS: You'll need the `@stomp/stompjs` and `sockjs-client` packages for WebSocket communication. Install them using npm:

```
npm install @stomp/stompjs sockjs-client
```

3. Create the React Component: Now, create a React component that will handle the WebSocket connection and messaging.

App.js

```
import React, { useState, useEffect, useRef } from 'react';  
import SockJS from 'sockjs-client';  
import { Client } from '@stomp/stompjs';  
  
const App = () => {  
  const [message, setMessage] = useState('');  
  const [name, setName] = useState('');  
  const stompClientRef = useRef(null);  
  
  useEffect(() => {  
    const socket = new SockJS('http://localhost:8080/ws');  
    const stompClient = new Client({  
      url: socket,  
      reconnectDelay: 5000,  
      onConnect: () => {  
        console.log('Connected to WebSocket');  
      },  
      onDisconnect: () => {  
        console.log('Disconnected from WebSocket');  
      },  
      onMessage: (message) => {  
        console.log('Received message:', message);  
        setMessage(message);  
      },  
    });  
    stompClientRef.current = stompClient;  
    stompClient.activate();  
  }, []);  
  
  return (

App

);  
};
```

Open in app ↗

Sign up

Sign in



```

    },
    onConnect: () => {
      console.log('Connected to WebSocket');
      stompClient.subscribe('/topic/greetings', (response) => {
        console.log('Received message:', response.body);
        setMessage(JSON.parse(response.body).content);
      });
    },
    onStompError: (frame) => {
      console.error('Broker reported error: ' + frame.headers['message']);
      console.error('Additional details: ' + frame.body);
    },
  });

  stompClient.activate();
  stompClientRef.current = stompClient;

  return () => {
    stompClient.deactivate();
  };
}, []);

const sendMessage = () => {
  const stompClient = stompClientRef.current;
  if (stompClient && stompClient.connected) {
    console.log('Sending message:', name);
    stompClient.publish({
      destination: '/app/hello',
      body: name,
    });
  } else {
    console.error('Stomp client is not connected');
  }
};

return (
  <div>
    <input
      type="text"
      placeholder="Enter your name"
      value={name}
      onChange={(e) => setName(e.target.value)}
    />
    <button onClick={sendMessage}>Send</button>
    <p>{message}</p>
  </div>
);
};

```

```
export default App;
```

In this React component:

- We use SockJS to create a WebSocket connection to our Spring Boot backend.
- We use the STOMP client to handle the WebSocket communication.
- The `useEffect` hook establishes the connection when the component mounts and cleans up when it unmounts.
- We subscribe to the `/topic/greetings` topic to receive messages from the server.
- The `sendMessage` function sends the name entered in the input field to the server.

How It Works Together

1. **Establish Connection:** When the React component mounts, it creates a WebSocket connection to the Spring Boot server using SockJS.
2. **Subscribe to Topic:** The client subscribes to the `/topic/greetings` topic to receive broadcast messages.
3. **Send Message:** When the user enters a name and clicks the send button, the `sendMessage` function sends the name to the server at the `/app/hello` endpoint.
4. **Handle Message:** The Spring Boot controller handles the message, creates a `Greeting` object, and sends it to the `/topic/greetings` topic.

5. Receive Message: The React client receives the greeting message and updates the state to display the message on the screen.

This setup allows real-time broadcast notifications to be sent from the server to all connected clients. It's a simple and effective way to add real-time features to your application.

I hope this explanation helps you understand the workflow of sending broadcast notifications using these technologies. If you have any questions, feel free to ask!



Written by Deshani Palliyaguruge

3 Followers · 8 Following

Follow

Undergraduate from the IT Faculty of the University of Moratuwa, Sri Lanka.

No responses yet



Write a response

What are your thoughts?

More from Deshani Palliyaguruge



Folder Structure
of a Spring Boot
Project



Deshani Palliyaguruge

Understanding the Folder Structure of a Spring Boot Project

Hello guys! Welcome to my first blog post.

May 17, 2024



2



How to Create a
REST API Using
Spring Boot



Deshani Palliyaguruge

How to Create a REST API Using Spring Boot

Hi guys, welcome.

May 22, 2024



1



See all from Deshani Palliyaguruge

Recommended from Medium



 Harendra

How I Am Using a Lifetime 100% Free Server

Get a server with 24 GB RAM + 4 CPU + 200 GB Storage + Always Free

★ Oct 26, 2024 🖱 9.4K 💬 170 📌




 Dilshara Hetti Arachchige

Getting Started with Lynx: A Next-Gen Cross-Platform Framework

The world of cross-platform app development just got a major shake-up. ByteDance, the...

Mar 7 🖱 226 💬 6 📌




 In Stackademic by Mohit Bajaj

How I Optimized a Spring Boot Application to Handle 1M...

Discover the exact techniques I used to scale a Spring Boot application from handling 50K...

★ Mar 2 🖱 240 💬 14 📌

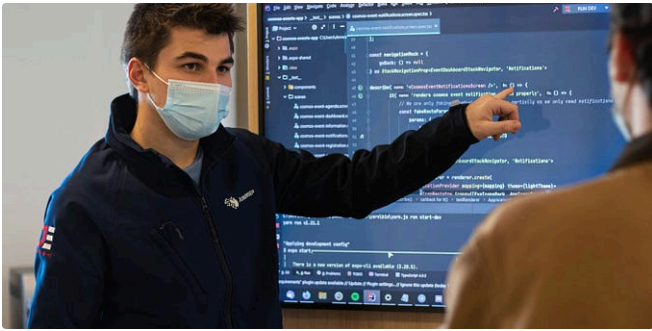


 In Javarevisited by Rasathurai Karan

Java's Funeral Has Been Announced.... 🐼 💻

Oh, Java is outdated! Java is too verbose! No one uses Java anymore!

★ Mar 7 🖱 1.1K 💬 65 📌



 Vinod Pal

How I Review Code As a Senior Developer For Better Results

I have been doing code reviews for quite some time and have become better at it. Fro...

★ Jan 25 🖱 1.6K 💬 39 

Spring Says Goodbye to @Autowired: Here's What to Use Instead



 Java Interview

Spring Says Goodbye to @Autowired: Here's What to Use...

Yes, starting with Spring Boot 3 and Spring Framework 6, Spring has been encouraging...

★ Feb 21 🖱 363 💬 16 

See more recommendations